Generating the Server Response: HTTP Status Codes

Topics in This Chapter

- Format of the HTTP response
- How to set status codes
- What the status codes are good for
- Shortcut methods for redirection and error pages
- A servlet that redirects users to browser-specific pages
- A front end to various search engines

Chapter 6

As we saw in the previous chapter, a request from a browser or other client consists of an HTTP command (usually `GET` or `POST`), zero or more request headers (one or more in HTTP 1.1, since `Host` is required), a blank line, and, only in the case of `POST` requests, some query data. A typical request looks like the following.

```
GET /servlet/SomeName HTTP/1.1
Host: ...
Header2: ...
...
HeaderN:
  (Blank Line)
```

When a Web server responds to a request, the response typically consists of a status line, some response headers, a blank line, and the document. A typical response looks like this:

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
...
</BODY></HTML>
```

The status line consists of the HTTP version (`HTTP/1.1` in the preceding example), a status code (an integer; `200` in the example), and a very short message corresponding to the status code (`OK` in the example). In most cases, the headers are optional except for `Content-Type`, which specifies the MIME type of the document that follows. Although most responses contain a document, some don't. For example, responses to `HEAD` requests should never include a document, and various status codes essentially indicate failure or redirection (and thus either don't include a document or include only a short error-message document).

Servlets can perform a variety of important tasks by manipulating the status line and the response headers. For example, they can forward the user to other sites; indicate that the attached document is an image, Adobe Acrobat file, or HTML file; tell the user that a password is required to access the document; and so forth. This chapter summarizes the most important status codes and describes what can be accomplished with them; the following chapter discusses the response headers.

6.1 Specifying Status Codes

As just described, the HTTP response status line consists of an HTTP version, a status code, and an associated message. Since the message is directly associated with the status code and the HTTP version is determined by the server, all a servlet needs to do is to set the status code. A code of 200 is set automatically, so servlets don't usually need to specify a status code at all. When they *do* want to, they use `response.setStatus`, `response.sendRedirect`, or `response.sendError`.

Setting Arbitrary Status Codes: setStatus

When you want to set an arbitrary status code, do so with the `setStatus` method of `HttpServletResponse`. If your response includes a special status code and a document, be sure to call `setStatus` *before* actually returning any of the content with the `PrintWriter`. The reason is that an HTTP response consists of the status line, one or more headers, a blank line, and the actual document, *in that order*. Servlets do not necessarily buffer the document, so you have to either set the status code before using the `PrintWriter` or carefully check that the buffer hasn't been flushed and content actually sent to the browser.

Core Approach

*Set status codes **before** sending any document content to the client.*

The `setStatus` method takes an `int` (the status code) as an argument, but instead of using explicit numbers, for readability and to avoid typos, use the constants defined in `HttpServletResponse`. The name of each constant is derived from the standard HTTP 1.1 message for each constant, all upper case with a prefix of `SC` (for *Status Code*) and spaces changed to underscores. Thus, since the message for 404 is Not Found, the equivalent constant in `HttpServletResponse` is `SC_NOT_FOUND`. There is one minor exception, however: the constant for code 302 is derived from the message defined by HTTP 1.0 (Moved Temporarily), not the HTTP 1.1 message (Found).

Setting 302 and 404 Status Codes: sendRedirect and sendError

Although the general method of setting status codes is simply to call `response.setStatus(int)`, there are two common cases for which a shortcut method in `HttpServletResponse` is provided. Just be aware that both of these methods throw `IOException`, whereas `setStatus` does not. Since the `doGet` and `doPost` methods already throw `IOException`, this difference only matters if you pass the response object to another method.

- **`public void sendRedirect(String url)`**
  The 302 status code directs the browser to connect to a new location. The `sendRedirect` method generates a 302 response along with a `Location` header giving the URL of the new document. Either an absolute or a relative URL is permitted; the system automatically translates relative URLs into absolute ones before putting them in the `Location` header.
- **`public void sendError(int code, String message)`**
  The 404 status code is used when no document is found on the server. The `sendError` method sends a status code (usually 404) along with a short message that is automatically formatted inside an HTML document and sent to the client.

Setting a status code does not necessarily mean that you omit the document. For example, although most servers automatically generate a small File Not Found message for 404 responses, a servlet might want to customize this response. Again, remember that if you do send output, you have to call `setStatus` or `sendError` *first*.

6.2 HTTP 1.1 Status Codes

In this section we describe the most important status codes available for use in servlets talking to HTTP 1.1 clients, along with the standard message associated with each code. A good understanding of these codes can dramatically increase the capabilities of your servlets, so you should at least skim the descriptions to see what options are at your disposal. You can come back for details when you are ready to use the capabilities.

The complete HTTP 1.1 specification is given in RFC 2616. In general, you can access RFCs online by going to http://www.rfc-editor.org/ and following the links to the latest RFC archive sites, but since this one came from the World Wide Web Consortium, you can just go to http://www.w3.org/Protocols/. Codes that are new in HTTP 1.1 are noted since some browsers support only HTTP 1.0. You should only send the new codes to clients that support HTTP 1.1, as verified by checking `request.getRequestProtocol`.

The rest of this section describes the specific status codes available in HTTP 1.1. These codes fall into five general categories:

- **100–199**
  Codes in the 100s are informational, indicating that the client should respond with some other action.
- **200–299**
  Values in the 200s signify that the request was successful.
- **300–399**
  Values in the 300s are used for files that have moved and usually include a `Location` header indicating the new address.
- **400–499**
  Values in the 400s indicate an error by the client.
- **500–599**
  Codes in the 500s signify an error by the server.

The constants in `HttpServletResponse` that represent the various codes are derived from the standard messages associated with the codes. In servlets, you usually refer to status codes only by means of these constants. For example, you would use `response.setStatus(response.SC_NO_CONTENT)` rather than `response.setStatus(204)`, since the latter is unclear to readers and is prone to typographical errors. However, you should note that servers are allowed to vary the messages slightly, and clients pay attention only to the numeric value. So, for example, you might see a server return a status line of `HTTP/1.1 200 Document Follows` instead of `HTTP/1.1 200 OK`.

100 (Continue)

If the server receives an `Expect` request header with a value of `100-continue`, it means that the client is asking if it can send an attached document in a follow-up request. In such a case, the server should either respond with status 100 (`SC_CONTINUE`) to tell the client to go ahead or use 417 (`SC_EXPECTATION_FAILED`) to tell the browser it won't accept the document. This status code is new in HTTP 1.1.

200 (OK)

A value of 200 (`SC_OK`) means that everything is fine; the document follows for `GET` and `POST` requests. This status is the default for servlets; if you don't use `setStatus`, you'll get 200.

202 (Accepted)

A value of 202 (`SC_ACCEPTED`) tells the client that the request is being acted upon but processing is not yet complete.

204 (No Content)

A status code of 204 (`SC_NO_CONTENT`) stipulates that the browser should continue to display the previous document because no new document is available. This behavior is useful if the user periodically reloads a page by pressing the Reload button and you can determine that the previous page is already up-to-date.

205 (Reset Content)

A value of 205 (`SC_RESET_CONTENT`) means that there is no new document but the browser should reset the document view. Thus, this status code is used to instruct browsers to clear form fields. It is new in HTTP 1.1.

301 (Moved Permanently)

The 301 (`SC_MOVED_PERMANENTLY`) status indicates that the requested document is elsewhere; the new URL for the document is given in the `Location` response header. Browsers should automatically follow the link to the new URL.

302 (Found)

This value is similar to 301, except that in principle the URL given by the `Location` header should be interpreted as a temporary replacement, not a permanent one. In practice, most browsers treat 301 and 302 identically. Note: in HTTP 1.0, the message was `Moved Temporarily` instead of `Found`, and the constant in `HttpServletResponse` is `SC_MOVED_TEMPORARILY`, not the expected `SC_FOUND`.

Core Note

The constant representing 302 is `SC_MOVED_TEMPORARILY`*, not* `SC_FOUND`*.*

Status code 302 is useful because browsers automatically follow the reference to the new URL given in the `Location` response header. Note that the browser reconnects to the new URL immediately; no intermediate output is displayed. This behavior distinguishes *redirects* from *refreshes* where an intermediate page is temporarily displayed (see the next chapter for details on the `Refresh` header). With redirects, another site, not the servlet itself, generates the results. So why use a servlet at all? Redirects are useful for the following tasks:

- **Computing destinations.** If you know the final destination for the user in advance, your hypertext link or HTML form could send the user directly there. But, if you need to look at the data before deciding where to obtain the necessary results, a redirection is useful. For example, you might want to send users to a standard site that gives information on stocks, but you need to look at the stock symbol before deciding whether to send them to the New York Stock Exchange, NASDAQ, or a non-U.S. site.
- **Tracking user behavior.** If you send users a page that contains a hypertext link to another site, you have no way to know if they actually click on the link. But perhaps this information is important in analyzing the usefulness of the different links you send them. So, instead of sending users the direct link, you can send them a link to your own site, where you can then record some information and then redirect them to the real site. For example, several search engines use this trick to determine which of the results they display are most popular.
- **Performing side effects.** What if you want to send users to a certain site but set a cookie on the user's browser first? No problem: return both a `Set-Cookie` response header (by means of `response.addCookie`—see Chapter 8) *and* a 302 status code (by means of `response.sendRedirect`).

The 302 status code is so useful, in fact, that there is a special method for it, `sendRedirect`. Using `response.sendRedirect(url)` has a couple of advantages over using `response.setStatus(response.SC_MOVED_TEMPORARILY)` and `response.setHeader("Location", url)`. First, it is shorter and easier. Second, with `sendRedirect`, the servlet automatically

builds a page containing the link to show to older browsers that don't automatically follow redirects. Finally, `sendRedirect` can handle relative URLs, automatically translating them into absolute ones.

Technically, browsers are supposed to automatically follow the redirection only if the original request was `GET`. For details, see the discussion of the 307 status code.

303 (See Other)

The 303 (`SC_SEE_OTHER`) status is similar to 301 and 302, except that if the original request was `POST`, the new document (given in the `Location` header) should be retrieved with `GET`. See status code 307. This code is new in HTTP 1.1.

304 (Not Modified)

When a client has a cached document, it can perform a conditional request by supplying an `If-Modified-Since` header to signify that it wants the document only if it has been changed since the specified date. A value of 304 (`SC_NOT_MODIFIED`) means that the cached version is up-to-date and the client should use it. Otherwise, the server should return the requested document with the normal (200) status code. Servlets normally should not set this status code directly. Instead, they should implement the `getLastModified` method and let the default `service` method handle conditional requests based upon this modification date. For an example, see the `LotteryNumbers` servlet in Section 3.6 (The Servlet Life Cycle).

307 (Temporary Redirect)

The rules for how a browser should handle a 307 status are identical to those for 302. The 307 value was added to HTTP 1.1 since many browsers erroneously follow the redirection on a 302 response even if the original message is a `POST`. Browsers are supposed to follow the redirection of a `POST` request only when they receive a 303 response status. This new status is intended to be unambiguously clear: follow redirected `GET` *and* `POST` requests in the case of 303 responses; follow redirected `GET` *but not* `POST` requests in the case of 307 responses. This status code is new in HTTP 1.1.

400 (Bad Request)

A 400 (`SC_BAD_REQUEST`) status indicates bad syntax in the client request.

401 (Unauthorized)

A value of 401 (`SC_UNAUTHORIZED`) signifies that the client tried to access a password-protected page but that the request did not have proper identifying

information in the `Authorization` header. The response must include a `WWW-Authenticate` header. For details, see the chapter on programmatic Web application security in Volume 2 of this book.

403 (Forbidden)

A status code of 403 (`SC_FORBIDDEN`) means that the server refuses to supply the resource, regardless of authorization. This status is often the result of bad file or directory permissions on the server.

404 (Not Found)

The infamous 404 (`SC_NOT_FOUND`) status tells the client that no resource could be found at that address. This value is the standard "no such page" response. It is such a common and useful response that there is a special method for it in the `HttpServletResponse` class: `sendError("message")`. The advantage of `sendError` over `setStatus` is that with `sendError`, the server automatically generates an error page showing the error message. 404 errors need not merely say "Sorry, the page cannot be found." Instead, they can give information on why the page couldn't be found or supply search boxes or alternative places to look. The sites at www.microsoft.com and www.ibm.com have particularly good examples of useful error pages (to see them, just make up a nonexistent URL at either site). In fact, there is an entire site dedicated to the good, the bad, the ugly, and the bizarre in 404 error messages: http://www.plinko.net/404/. We find http://www.plinko.net/404/links.asp?type=cat&key=13 (amusing 404 error messages) particularly funny.

Unfortunately, however, the default behavior of Internet Explorer in version 5 and later is to ignore the error page you send back and to display its own static (and relatively useless) error message, even though doing so explicitly contradicts the HTTP specification. To turn off this setting, go to the Tools menu, select Internet Options, choose the Advanced tab, and make sure the "Show friendly HTTP error messages" box is *not* checked. Regrettably, few users are aware of this setting, so this "feature" prevents most users of Internet Explorer from seeing any informative messages you return. Other major browsers and version 4 of Internet Explorer properly display server-generated error pages.

Core Warning

By default, Internet Explorer versions 5 and later improperly ignore server-generated error pages.

Fortunately, it is relatively uncommon for individual servlets to build their own 404 error pages. A more common approach is to set up error pages for an entire Web application; see Section 2.11 (Web Applications: A Preview) for details.

405 (Method Not Allowed)

A 405 (`SC_METHOD_NOT_ALLOWED`) value signifies that the request method (`GET`, `POST`, `HEAD`, `PUT`, `DELETE`, etc.) was not allowed for this particular resource. This status code is new in HTTP 1.1.

415 (Unsupported Media Type)

A value of 415 (`SC_UNSUPPORTED_MEDIA_TYPE`) means that the request had an attached document of a type the server doesn't know how to handle. This status code is new in HTTP 1.1.

417 (Expectation Failed)

If the server receives an `Expect` request header with a value of `100-continue`, it means that the client is asking if it can send an attached document in a follow-up request. In such a case, the server should either respond with this status (417) to tell the browser it won't accept the document or use 100 (`SC_CONTINUE`) to tell the client to go ahead. This status code is new in HTTP 1.1.

500 (Internal Server Error)

500 (`SC_INTERNAL_SERVER_ERROR`) is the generic "server is confused" status code. It often results from CGI programs or (heaven forbid!) servlets that crash or return improperly formatted headers.

501 (Not Implemented)

The 501 (`SC_NOT_IMPLEMENTED`) status notifies the client that the server doesn't support the functionality to fulfill the request. It is used, for example, when the client issues a command like `PUT` that the server doesn't support.

503 (Service Unavailable)

A status code of 503 (`SC_SERVICE_UNAVAILABLE`) signifies that the server cannot respond because of maintenance or overloading. For example, a servlet might return this header if some thread or database connection pool is currently full. The server can supply a `Retry-After` header to tell the client when to try again.

505 (HTTP Version Not Supported)

The 505 (`SC_HTTP_VERSION_NOT_SUPPORTED`) code means that the server doesn't support the version of HTTP named in the request line. This status code is new in HTTP 1.1.

6.3 A Servlet That Redirects Users to Browser-Specific Pages

Recall from Chapter 5 that the `User-Agent` request header designates the specific browser (or cell phone or other client) making the request. Recall further that most major browsers contain the string `Mozilla` in their `User-Agent` header, but only Microsoft Internet Explorer contains the string `MSIE`.

Listing 6.1 shows a servlet that makes use of this fact to send Internet Explorer users to the Netscape home page, and all other users to the Microsoft home page. The servlet accomplishes this task by using the `sendRedirect` method to send a 302 status code and a `Location` response header to the browser. Figures 6–1 and 6–2 show results for Internet Explorer and Netscape, respectively.

Listing 6.1 WrongDestination.java

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Servlet that sends IE users to the Netscape home page and
 *  Netscape (and all other) users to the Microsoft home page.
 */

public class WrongDestination extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String userAgent = request.getHeader("User-Agent");
    if ((userAgent != null) &&
        (userAgent.indexOf("MSIE") != -1)) {
      response.sendRedirect("http://home.netscape.com");
    } else {
      response.sendRedirect("http://www.microsoft.com");
    }
  }
}
```

Figure 6–1 Result of http://*host*/servlet/coreservlets.WrongDestination in Internet Explorer.

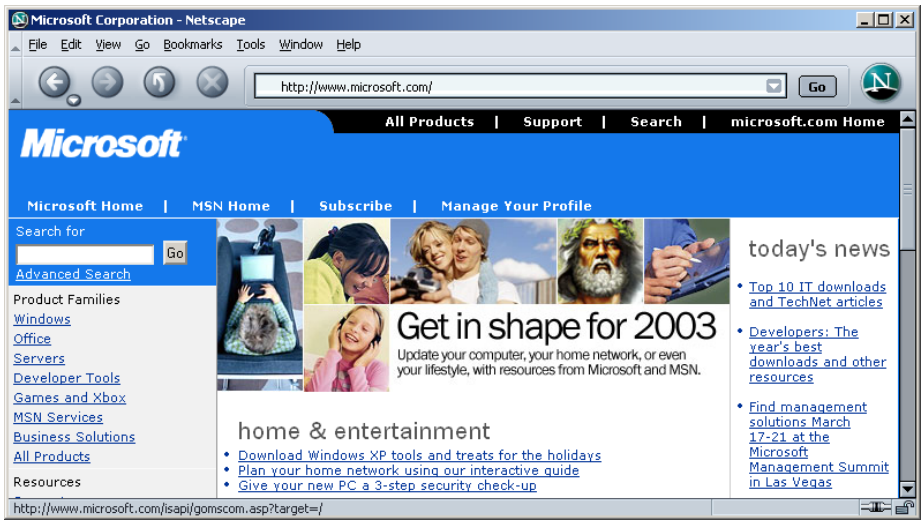

Figure 6–2 Result of http://*host*/servlet/coreservlets.WrongDestination in Netscape.

6.4 A Front End to Various Search Engines

Suppose that you want to make a "one-stop searching" site that lets users search any of the most popular search engines without having to remember many different URLs. You want to let users enter a query, select the search engine, and then send them to that search engine's results page for that query. If users omit the search keywords or fail to select a search engine, you have no site to redirect them to, so you want to display an error page informing them of this fact.

Listing 6.2 (`SearchEngines.java`) presents a servlet that accomplishes these tasks by making use of the 302 (`Found`) and 404 (`Not Found`) status codes—the two most common status codes other than 200. The 302 code is set by the shorthand `sendRedirect` method of `HttpServletResponse`, and 404 is specified by `sendError`.

In this application, a servlet builds an HTML form (see Figure 6–3 and the source code in Listing 6.5) that displays a page to let the user specify a search string and select the search engine to use. When the form is submitted, the servlet extracts those two parameters, constructs a URL with the parameters embedded in a way appropriate to the search engine selected (see the `SearchSpec` and `SearchUtilities` classes of Listings 6.3 and 6.4), and redirects the user to that URL (see Figure 6–4). If the user fails to choose a search engine or specify search terms, an error page informs the client of this fact (see Figure 6–5, but recall warnings about Internet Explorer under the 404 status code in the previous section).

Listing 6.2 SearchEngines.java

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;

/** Servlet that takes a search string and a search
 *  engine name, sending the query to
 *  that search engine. Illustrates manipulating
 *  the response status code. It sends a 302 response
 *  (via sendRedirect) if it gets a known search engine,
 *  and sends a 404 response (via sendError) otherwise.
 */
```

Listing 6.2 SearchEngines.java *(continued)*

```
public class SearchEngines extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String searchString = request.getParameter("searchString");
    if ((searchString == null) ||
        (searchString.length() == 0)) {
      reportProblem(response, "Missing search string");
      return;
    }
    // The URLEncoder changes spaces to "+" signs and other
    // non-alphanumeric characters to "%XY", where XY is the
    // hex value of the ASCII (or ISO Latin-1) character.
    // Browsers always URL-encode form values, and the
    // getParameter method decodes automatically. But since
    // we're just passing this on to another server, we need to
    // re-encode it to avoid characters that are illegal in
    // URLs. Also note that JDK 1.4 introduced a two-argument
    // version of URLEncoder.encode and deprecated the one-arg
    // version. However, since version 2.3 of the servlet spec
    // mandates only the Java 2 Platform (JDK 1.2 or later),
    // we stick with the one-arg version for portability.
    searchString = URLEncoder.encode(searchString);

    String searchEngineName =
      request.getParameter("searchEngine");
    if ((searchEngineName == null) ||
        (searchEngineName.length() == 0)) {
      reportProblem(response, "Missing search engine name");
      return;
    }
    String searchURL =
      SearchUtilities.makeURL(searchEngineName, searchString);
    if (searchURL != null) {
      response.sendRedirect(searchURL);
    } else {
      reportProblem(response, "Unrecognized search engine");
    }
  }

  private void reportProblem(HttpServletResponse response,
                             String message)
      throws IOException {
    response.sendError(response.SC_NOT_FOUND, message);
  }
}
```

Listing 6.3 SearchSpec.java

```
package coreservlets;

/** Small class that encapsulates how to construct a
 *  search string for a particular search engine.
 */

public class SearchSpec {
  private String name, baseURL;

  public SearchSpec(String name,
                    String baseURL) {
    this.name = name;
    this.baseURL = baseURL;
  }

  /** Builds a URL for the results page by simply concatenating
   *  the base URL (http://...?someVar=") with the URL-encoded
   *  search string (jsp+training).
   */

  public String makeURL(String searchString) {
    return(baseURL + searchString);
  }

  public String getName() {
    return(name);
  }
}
```

Listing 6.4 SearchUtilities.java

```
package coreservlets;

/** Utility with static method to build a URL for any
 *  of the most popular search engines.
 */

public class SearchUtilities {
  private static SearchSpec[] commonSpecs =
    { new SearchSpec("Google",
                     "http://www.google.com/search?q="),
      new SearchSpec("AllTheWeb",
                     "http://www.alltheweb.com/search?q="),
```

Listing 6.4 SearchUtilities.java *(continued)*

```
      new SearchSpec("Yahoo",
                     "http://search.yahoo.com/bin/search?p="),
      new SearchSpec("AltaVista",
                     "http://www.altavista.com/web/results?q="),
      new SearchSpec("Lycos",
                     "search.lycos.com/default.asp?query="),
      new SearchSpec("HotBot",
                     "http://hotbot.com/default.asp?query="),
      new SearchSpec("MSN",
                     "http://search.msn.com/results.asp?q="),
    };

  public static SearchSpec[] getCommonSpecs() {
    return(commonSpecs);
  }

  /** Given a search engine name and a search string, builds
   *  a URL for the results page of that search engine
   *  for that query. Returns null if the search engine name
   *  is not one of the ones it knows about.
   */

  public static String makeURL(String searchEngineName,
                               String searchString) {
    SearchSpec[] searchSpecs = getCommonSpecs();
    String searchURL = null;
    for(int i=0; i<searchSpecs.length; i++) {
      SearchSpec spec = searchSpecs[i];
      if (spec.getName().equalsIgnoreCase(searchEngineName)) {
        searchURL = spec.makeURL(searchString);
        break;
      }
    }
    return(searchURL);
  }
}
```

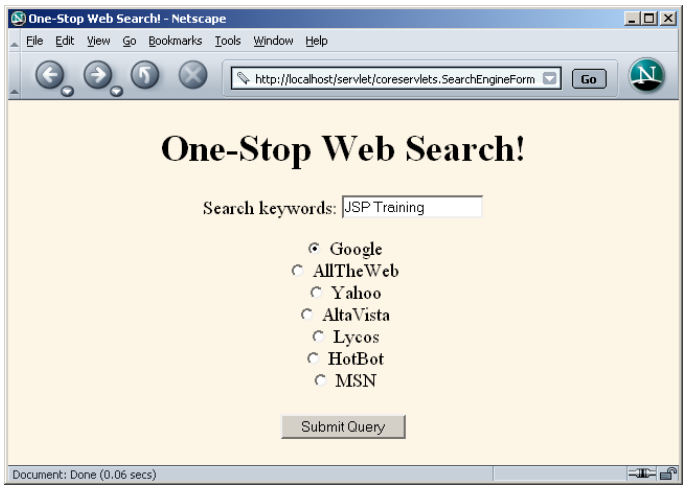

Figure 6–3 Front end to the `SearchEngines` servlet. See Listing 6.5 for the source code.

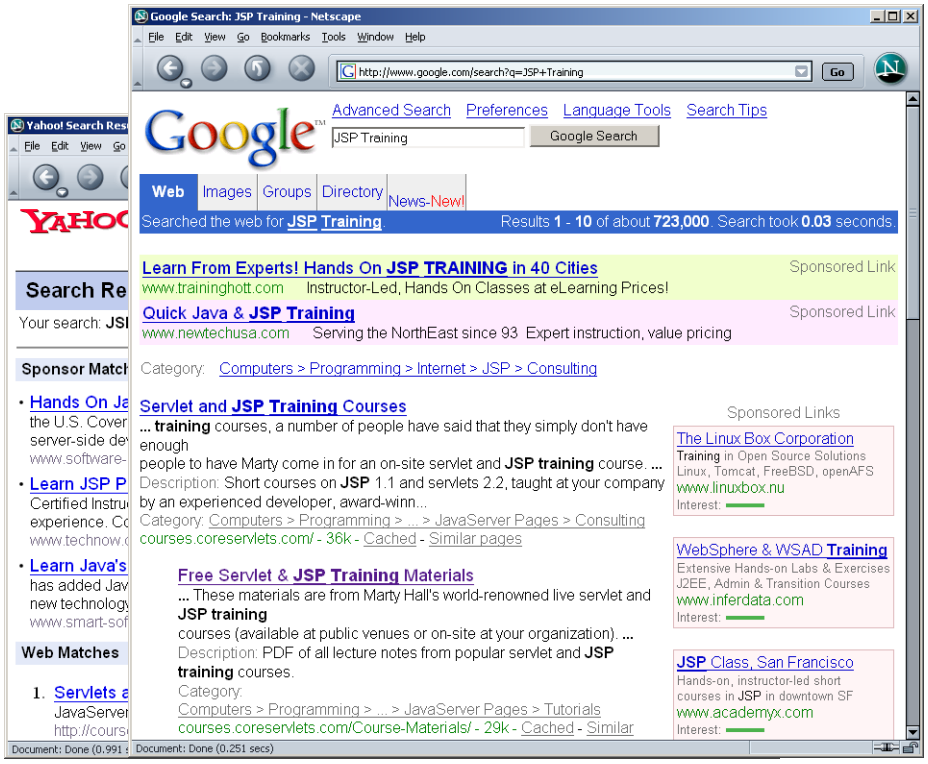

Figure 6–4 Results of the `SearchEngines` servlet when the form of Figure 6–3 is submitted. Although the form is submitted to the `SearchEngines` servlet, that servlet generates no output and the end user sees only the result of the redirection.

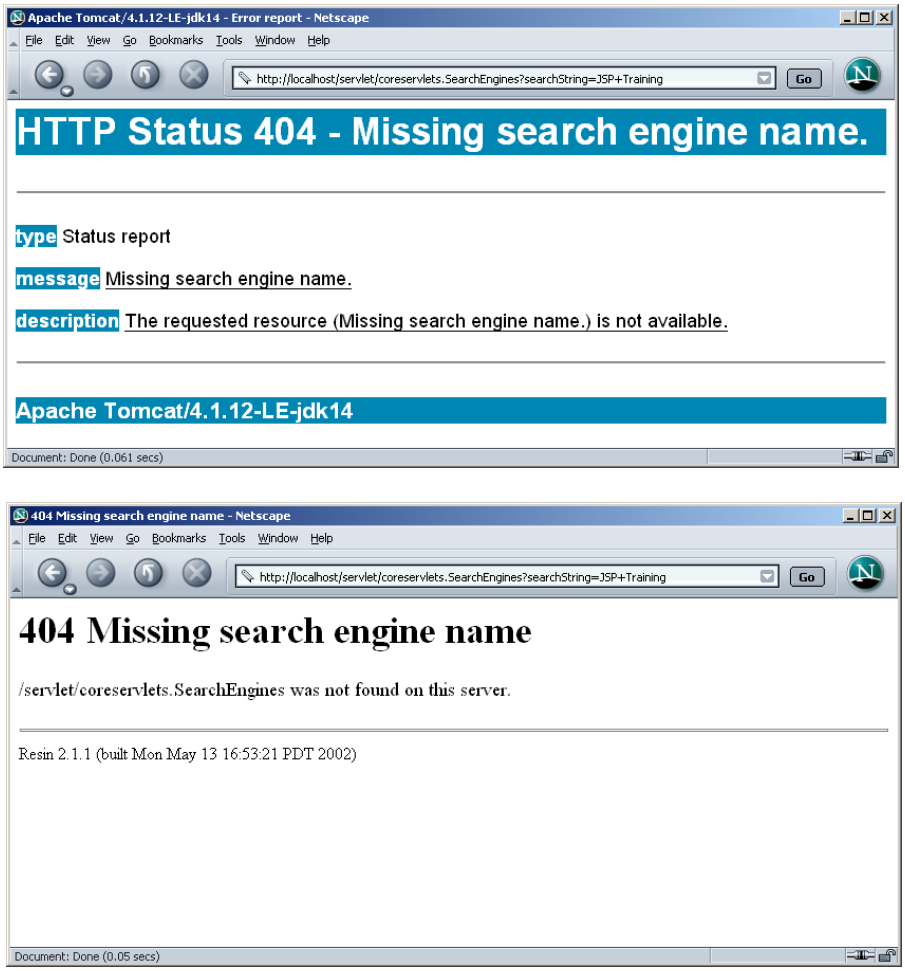

Figure 6–5 Results of the `SearchEngines` servlet upon submission of a form that has no search engine specified. These results are for Tomcat 4.1 and Resin 4.0; results will vary slightly among servers and will incorrectly omit the "Missing search string" message in JRun 4. In Internet Explorer, you must modify the browser settings as described in the previous section (see the 404 entry) to see the error message.

Listing 6.5 SearchEngineForm.java

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Servlet that builds the HTML form that gathers input
 *  for the search engine servlet. This servlet first
 *  displays a textfield for the search query, then looks up
 *  the search engine names known to SearchUtilities and
 *  displays a list of radio buttons, one for each search
 *  engine.
 */

public class SearchEngineForm extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "One-Stop Web Search!";
    String actionURL = "/servlet/coreservlets.SearchEngines";
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println
      (docType +
       "<HTML>\n" +
       "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
       "<BODY BGCOLOR=\"#FDF5E6\">\n" +
       "<CENTER>\n" +
       "<H1>" + title + "</H1>\n" +
       "<FORM ACTION=\"" + actionURL + "\">\n" +
       "  Search keywords: \n" +
       "  <INPUT TYPE=\"TEXT\" NAME=\"searchString\"><P>\n");
    SearchSpec[] specs = SearchUtilities.getCommonSpecs();
    for(int i=0; i<specs.length; i++) {
      String searchEngineName = specs[i].getName();
      out.println("<INPUT TYPE=\"RADIO\" " +
                  "NAME=\"searchEngine\" " +
                  "VALUE=\"" + searchEngineName + "\">\n");
      out.println(searchEngineName + "<BR>\n");
    }
    out.println
      ("<BR>  <INPUT TYPE=\"SUBMIT\">\n" +
       "</FORM>\n" +
       "</CENTER></BODY></HTML>");
  }
}
```